VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY

FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
ÚSTAV INFORMAČNÍCH SYSTÉMŮ

FACULTY OF INFORMATION TECHNOLOGY
DEPARTMENT OF INFORMATION SYSTEMS

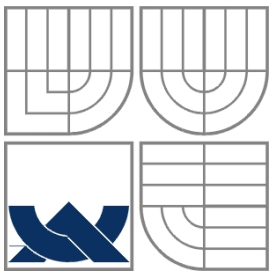
ZÍSKÁVÁNÍ ZNALOSTÍ Z DATABÁZÍ

BAKALÁŘSKÁ PRÁCE
BACHELOR´S THESIS

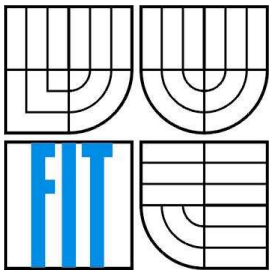
AUTOR PRÁCE
AUTHOR

ONDŘEJ JAROŠ

BRNO 2008



VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY



FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
ÚSTAV INFORMAČNÍCH SYSTÉMŮ

FACULTY OF INFORMATION TECHNOLOGY
DEPARTMENT OF INFORMATION SYSTEMS

ZÍSKÁVÁNÍ ZNALOSTÍ Z DATABÁZÍ

KNOWLEDGE DATA DISCOVERY

BAKALÁŘSKÁ PRÁCE
BACHELOR´S THESIS

AUTOR PRÁCE
AUTHOR

ONDŘEJ JAROŠ

VEDOUCÍ PRÁCE
SUPERVISOR

Ing. PAVEL JURKA

BRNO 2008

Abstrakt

Tato bakalářská práce se zabývá problematikou získávání znalostí z databází. Jsou zde podrobně probrány metody pro dolování dat z databází. Konkrétně se tato práce zaměřuje na metodu získávání znalostí pomocí Asociačních pravidel a to za pomocí algoritmu Apriori. Cílem práce bylo implementovat vybranou metodu a její funkčnost ověřit na vybraném vzorku dat. Aplikace je implementována v programovacím jazyce Java a vzorek dat, nad kterými metoda pracuje je uložen v souboru XML. Nejedná se tedy o klasické získávání znalostí z databáze, nýbrž z XML souboru. Informace pro spuštění dolovací úlohy jsou načítány z konfiguračního(systémového) XML dokumentu. Získané znalosti jsou poté ukládány do systémového XML dokumentu.

Klíčová slova

Získávání znalostí z databází,znalosti, dolování dat, asociace, pravidla, Asociační pravidla, Apriori, Apriori podmínka, Java, XML

Abstrakt

This Bachelor work describes knowledge data discovery. There are in detail described methods of data mining in this text. This work is focusing principally on Associative rules method of knowledge data discovery, applying Apriory algorithm. The purpose of this work was to implement chosen method and verify its functionality on particular sample of data. Application is implemented in Java programming language and the sample of data, used for our method, is saved in XML file. Application doesnt work with classic database, but with XML file. Information needed to run the program is loaded from input (system) XML document. Discovered data are saved to system XML document.

Keywords

Knowledge data discovery, knowledge, Data mining, association, rules, Association rules, Apriori, Apriori term, Java, XML.

Citace

Ondřej Jaroš: Získávání znalostí z databází. Brno, 2008, bakalářská práce, FIT VUT v Brně.

Získávání znalostí z databází

Prohlášení

Prohlašuji, že jsem tuto bakalářskou práci vypracoval samostatně pod vedením pana Ing. Pavla Jurky
Uvedl jsem všechny literární prameny a publikace, ze kterých jsem čerpal.

……
Ondřej Jaroš
23.4.2008

Poděkování

Děkuji vedoucímu bakalářské práce panu Ing. Pavlu Jurkovi za poskytnutí konzultací a odborných rad pří řešení problémů

Obsah

Úvod

V současné době dochází k velkému nárůstu objemu uložených dat v elektronické podobě. Díky široké dostupnosti a snaze přeměnit tato data na užitečné informace a znalosti si obor Získávání znalostí z databází získal brzy značnou pozornost. Tyto data, jsou pak často využita pro vytváření statistik, analýz, průzkumů trhu apod. Ovšem dnešní data neobsahují pouze užitečná data, která by se dala lehko zpracovat obvyklými postupy. Data obsahují nekvalitní či neúplné vzorky dat nazývané též „šum“. Proto bylo vyvinuto několik technik, které jsou součástí celého systému pro zpracování velkého objemu dat. Tyto metody musí být schopné spolupracovat s různými úložišti dat, jako jsou například databáze(relační databáze, XML dokumenty, …), datové sklady apod.

V této práci se věnuji zpracování dat uložených v XML souborech. V úvodní kapitole popisuji problematiku získávání znalostí za databází, jaké má části, jaká data jsou pro získávání znalostí vhodná.

V druhé kapitole se zaměřuji na problematiku týkající se asociačních pravidel pro získávání znalostí. Jejich rozdělení a popis jednotlivých metod. Dále je v této kapitole uveden algoritmus Apriori, který je implementován v aplikaci, která je součástí této práce.

Protože tato práce se zabývá dolováním znalostí z XML souborů, je ve třetí kapitole popis XML standartu společně s pravidly pro psaní XML souborů.

Ve čtvrté kapitole se zabývám návrhem koncepce aplikace. Je zde stručně popsán formát uložených dat v XML a formát vstupního konfiguračního souboru.

Pátá kapitola obsahuje podrobný popis implementace aplikace. Podrobný popis návazností a funkčností jednotlivých metod, ze kterých je aplikace tvořena.

V předposlední kapitole je uvedeno ověřování funkčnosti aplikace na reálných datech. Je zde uveden popis testovaných dat, podrobný popis experimentů a výsledků, které z těchto experimentů byly dosaženy.

Poslední kapitolou je samotný závěr této práce. Zde jsou shrnuty poznatky a výsledky této práce.

1 Kapitola 1.

1.1 Získávání znalostí z databází

1.1.1 Úvod

Pojem jakým je Získávání znalostí z databází asi nejlépe vystihuje definice podle [2]: „*Jde o dolování zajímavých, netriviálních a potenciálně užitečných dat*“. Tedy tyto informace se nedají získat jednoduchým dotazem, například prostřednictvím jazyka SQL a cílová data jsou skryta v datech. Je potřeba taková data nejprve nalézt a poté je můžeme použít, jako podklady pro různá rozhodnutí. Další významnou charakteristikou pro získávání znalostí je, že jsou podporovány v mnoha disciplínách, jež jsou nejvíce zastoupeny obory statistik, obchodních analýz a oboru umělé inteligence. Dále bych chtěl upozornit na rozdíl mezi českými názvy Získávání znalostí z databází (Knowledge Discovery in Databases, KDD) a Dolování dat (Data mining, DM). Získávání znalostí z databází je název zahrnující celý projekt získávání užitečných dat, kdežto Dolování dat je pouze jedna z jeho součástí. I když se v dnešní době uplatňují oba dva názvy pro označení celého projektu získávání znalostí.

1.1.2 Proces získávání znalostí

Proces získávání znalostí z databází je posloupnost několika na sebe navazujících kroků pomocí, kterých získáme potřebné výsledky. Většinou se některé kroky v určitých iteracích opakují. Celý proces lze rozdělit na tyto části:

1. Čištění a integrace dat

Velmi často se tyto dva první kroky spojují dohromady. Jednak proto, že vyčištěná data je potřeba někam ukládat a proto že, nekonzistence často vzniká spojením dat z různých zdrojů. Cílem je

vyčistění dat , odstranění šumu, nekonzistentních dat a vypořádání se s neúplnými daty. Dalším cílem je integrovat všechna data pocházející od různých zdrojů.

2. Výběr dat

V tomto kroku se vybírají data, která jsou pro získání znalostí relevantní (např. u relační databáze vybereme správné sloupce).

3. Transformace dat

Zde je potřeba vybraná data zpracovat do podoby vhodné pro danou dolovací metodu. Například sumarizace, nebo agregace. Zároveň je tímto krokem ukončeno tzv. Předzpracování dat, jenž často souhrnně označujeme, jako první krok získávání znalostí z databází.

4. Dolování dat

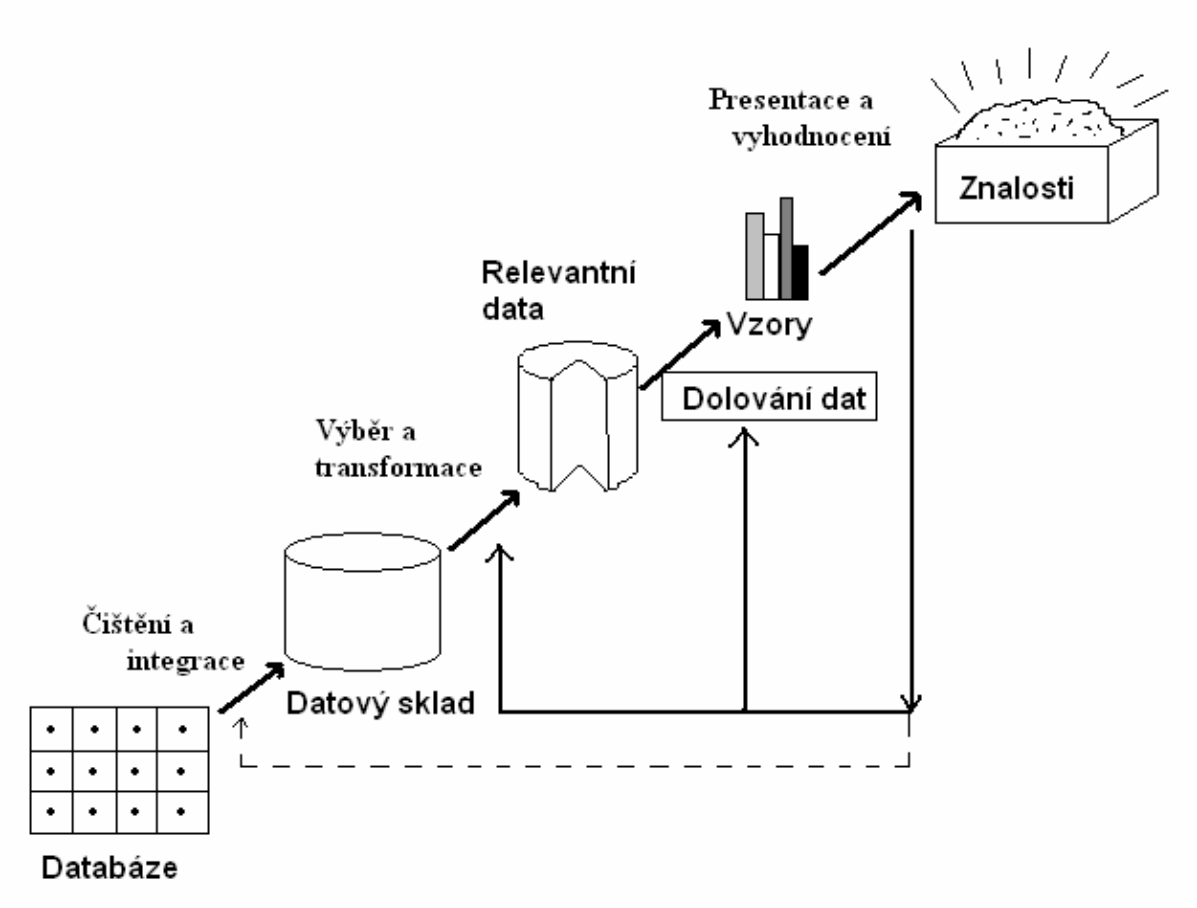
Jádro celého procesu. Za použití vhodné metody a algoritmu pro daný účel získáme z předzpracovaných dat požadované znalosti. Jedná se o nejdůležitější krok v celém procesu.

6. Hodnocení metod a vzorů

Cílem tohoto kroku je identifikace zajímavých a užitečných vzorů dat získaných z předcházejícího kroku.

7. Presentace znalostí

Posledním krokem je samotná presentace získaných znalostí srozumitelnou formou koncovému uživateli.



Obr 1.2 – Proces získávání znalostí

1.1.3 Druhy dat vhodná pro dolování

Znalosti lze teoreticky dolovat z jakýchkoliv druhů databází. Nejčastějším úložištěm dat jsou relační databáze, dále pak datové sklady, transakční databáze, objektově-relační databáze, temporální databáze, proudy dat, web a další.

V této práci se zaměřuje na textové databáze především pak na dolování dat z XML dokumentů. XML dokument se ve značné míře liší od relační databáze především uložením dat. Data zde nebývají často uložena v několik samostatných tabulkách, ale pouze v jedné velké tabulce. Každý záznam je jednoznačně identifikovatelný a obsahuje několik atributů.

1.1.4 Typy dolovacích úloh

V podstatě jde o to, jaký druh modelu dat se snažíme z dat získat za použití vhodné metody. Tento krok se provádí v samém jádru celého procesu a to v dolování dat. Tyto typy dolovacích úloh můžeme rozdělit do dvou základních skupin.

- **Deskriptivní**
  Tyto dolovací úlohy charakterizují obecné vlastnosti analyzovaných dat.

- **Prediktivní**
  Využívají analýzy stávajících dat, aby mohli předpokládat chování budoucích dat. Typickým představitelem těchto typů úloh je klasifikace.

Nyní popíšeme jednotlivé základní typy dolovacích úloh.

Popis konceptu/třídy
Jednotlivá data mohou být sdružována s určitou třídou nebo konceptem. Takový popis můžeme získat jedním ze dvou způsobů:

Charakterizace dat – jedná se o sumarizaci obecných vlastností zkoumaných dat.

Diskriminace dat – popisuje rozdíl mezi obecnými vlastnostmi cílové třídy a obecnými vlastnostmi jedné, nebo několika rozdílových tříd.

Asociační pravidla
Hledání frekventovaných vzorů dat vede k odhalení zajímavým asociací a korelací. Použitím asociačních pravidel dostáváme implikaci typu X => Y, které říká, je li v transakci obsažen prvek X je obsažen s velkou pravděpodobností i prvek Y. Společně s pravidly se udávají i atributy *podpora* a *spolehlivost*. Výsledkem takových to asociací může být přiklad:

věk(X, ´50 -59´) ∧ váha(X, ´90-99´) => tlak(X, ´vysoký´) [podpora=1% , spolehlivost=50%]

kde X je proměnná reprezentující pacienta a pravidlo říká, že pacient, který má věk mezi 50 až 59 a zároveň váhu v rozmezí 90 až 99 kg má tendenci mít vysoký tlak. Míry podpory a spolehlivosti udávají významnost zastoupení příslušného frekventovaného vzoru v analyzovaných datech. Více o asociačních pravidlech v následující kapitole 2.

Klasifikace a predikce

Tyto typy úloh se řadí do skupiny prediktivních úloh. Cílem klasifikace je nalézt taková pravidla a metody dat, která popisují jednotlivá data a zároveň je od sebe odlišují. Takto klasifikovaná data, se pak dají použít pro predikci třídy objektů dat, jejichž třída je neznáma. Celý proces klasifikace se skládá ze tří základních kroků:

Trénování – za použití trénovaní množiny je vytvořen klasifikační model
Testování – za použití testovacích dat se provede hodnocení vytvořeného modulu
Aplikace – použití vytvořeného modulu pro klasifikaci jehož třídu neznáme

Shluková analýza

Shluková analýza se používá stejně jako klasifikace pro rozdělení vzorků dat do tříd, ale na rozdíl od klasifikace nejsou předem známy žádné modely, nebo rozdílové třídy podle kterých by mohla být data rozdělena. Cílem je tedy najít data s co největší podobností uvnitř shluku a naopak s co minimální podobností mezi jednotlivými shluky. Objekty se shlukují na základě maximalizace podobností objektů téže třídy a minimalizace objektů různých tříd. Výsledkem jsou třídy podobné *shlukům*.

Analýza odlehlých hodnot

Jinak též nazývané dolovaní odlehlých hodnot. Zajímavé vzory dat mohou představovat i takové hodnoty, které se v analyzovaných datech nevyskytují často, ale takové, které se od ostatních dat určitým způsobem odlišují. Takové objekty se nazývají *odlehlé hodnoty*, proto analýza odlehlých hodnot. Těchto metod se velice často využívá při odhalování podvodných machinací například s kreditními kartami. Například je li proveden neobvykle velký nákup na rozdíl od jiných, či na neobvyklém místě a podobně.

Analýza evoluce

Analýza evoluce popisuje a modeluje pravidelnosti a trendy u objektů, jejichž chování se mění v čase. I u takových to dat, lze použít některou z předešlých metod. Konkrétně pro tuto metodu existují speciálně zaměřené úlohy.

2 Asociační pravidla

2.1 Získávání asociačních pravidel – základní pojmy

2.1.1 Co jsou to asociační pravidla?

Asociační pravidla jsou v podstatě implikace tvar X ⇒Y. Tato implikace vyjadřuje to, že pokud je v dané množině hodnot obsažena položka X, je v ní také s velkou pravděpodobností obsažena položka Y. podstatě jde o vyhledávání zajímavých asociací, souvislostí, o kterých nevíme a můžeme je považovat za potenciálně užitečné, v jisté databázi dat. Asociační pravidla si našla velká uplatnění především v oblasti marketingu. Jako příklad můžeme uvést tvrzení: Koupí-li si zákazník X, koupí si i s velkou pravděpodobností zboží Y. Podle [2] a [4].

2.1.2 Formální definice asociačních pravidel

Nechť I = $\{i_1, i_2\ i_3, \ldots\}$ je množina všech položek, D je množina transakcí T, kde každá transakce T je množina položek taková, že T ⊆ I. Každá transakce T má svůj vlastní jedinečný identifikátor TID. Nechť X je množina položek, pak říkáme, že transakce T obsahuje položky X tehdy a jen tehdy, pokud X ⊆ T. Asociační pravidlo je implikace tvaru X ⇒Y, kde X ⊆ T, Y ⊆ T a X ∩ Y = ∅.

Pravidlo X ⇒ Y má podporu (supp) v množině transakcí D, jestliže supp% transakcí v D obsahuje množinu položek X ∪ Y.

Pravidlo X ⇒ Y má v množině D spolehlivost (conf) c, jestliže c% transakcí obsahuje položku X obsahuje také položky Y.

Pomocí těchto dvou parametrů (conf, supp), lze zjistit jak významné pravidlo jsme z databáze získaly, neboli jak je pro nás tento vztah zajímavý. Na začátku práce s asociačními pravidly je potřeba si určit nějakou dolní mez, abychom mohli vyřadit méně zajímavá pravidla. Většinou se bere minimální mez podpory a spolehlivosti. [2], [4]

Obecně lze získávání asociačních pravidel rozdělit do dvou fází:

1) V první fázi se získají tzv. *frekventované množiny* prvků , čili množiny, které mají stanovenou podporu větší než je nějaká požadovaná mez. Ostatní se neberou při dalším používání v potaz. Tento krok je iterativní tzn., že se napřed vytvoří jednoprvkové množiny prvků, které se vyskytuji v množině transakcí, dále se z nich vytvoří dvou prvkové atd.

2) Ve druhém kroku se z frekventovaných množin získají silná asociační pravidla, což jsou množiny splňující požadovanou minimální spolehlivost. Ostatní se neberou v potaz. Výsledkem je tedy množina silných asociačních pravidel, která mohou být potencionálně zajímavá.

2.1.3 Různé typy asociačních pravidel

Asociační pravidla mohou být klasifikována podle následujících kritérii:

Podle typu hodnot v pravidlech
Pokud nás zajímá pouze to zda-li je nějaký prvek obsažen, nebo ne, jedná se o booleovská asociační pravidla, pokud nás však zajímá asociace mezi kvantitativními položkami nebo atributy jedná se o kvantitativní asociační pravidla.

Podle dimenzí obsažených v pravidlech
Jedno-dimenzionální pravidla jsou například booleovská pravidla, protože obsahují pouze jednu dimenzi, u které nás zajímá pouze to, zda li je v prvek obsažen v dolovaných datech. U více dimenzionálních pravidel je už více pravidel, jedná se například o pravidlo typu:

$$věk(X, ´50\text{-}59´) \wedge váha(X, ´90\text{-}99´) \Rightarrow tlak(X, ´vysoký´),$$

které obsahuje 3 dimenze věk, výška, tlak.

Podle dalších rozšíření asociačních pravidel
Dalšími rozšířeními asociačních pravidel mohou být například porovnání kritérii podle úrovně abstrakce v pravidlech, korelační analýza, uzavřené množiny nebo maximální vzory.

2.2 Dolování jednoúrovňových booleovských asociačních pravidel

U tohoto typu asociačních pravidel nás zajímá jen to, zda li je prvek v množině obsažen či nikoli. Z tohoto hlediska se jedná o nejjednodušší typ asociačních pravidel. Nejznámějším algoritmem používaným pro jednoúrovňová booleovská pravidla je *algoritmus Apriori*, nebo jeho modifikace.

2.2.1 Algoritmus Apriori

Algoritmus apriori slouží pro získávání frekventovaných množin položek pro booleovská asociační pravidla. Algoritmus využívá znalostí o dříve získané vlastnosti frekventovaných množin. Ty jsou pak použity při generování další (k+1) množiny. Pro každé iterativní generování je, tak potřeba procházet databázi. Proto algoritmus využívá pro vyšší efektivitu tzv. Apriori podmínku. Tato podmínka říká, že každá podmnožina frekventovaných množin musí být také frekventovaná. Není li tomu tak, množina se dále neuvažuje. Protože přidáním jednoho prvku k frekventované množině nemůže způsobit, že by jeho podpora vzrostla.

Nyní si ukážeme samotný algoritmus Apriori s jeho popisem. Podle [2] a [4].

Vstupem do algoritmu je databáze transakcí D a hodnoty minimální podpory min_supp
Výstupem je L – frekventované množiny v databázi D
Algoritmus:

1) L_1 = nalezni_frekventované_1-množiny(D);
2) **for**$(k=2; L_{k-1} \neq \varnothing ; k++)$**{**
3) C_k=*apriori_gen*(L_{k-1}, *min_supp*); *//funkce pro generování nových kandidátů*
4) **for each** transakce $t \in D${ *//průchod databázi pro zjištění počtu výskytů*
5) C_t = *subset(Ck, f);* *//kandidátské množiny obsažené v transakci t*
6) **for each** kandidáti $c \in C_t$
7) *c.počet_výskytů++;*
8) }
9) $L_k = \{c \in C_k \mid c.počet\_výskytů \geq$ min_supp$\}$;
10) }
11) **return** $L=U_k L_k$;

Metoda: apriori_gen( );

Algoritmus:

1) **for each** množina $l_1 \in L_{k-1}$

2) **for each** množina $l_2 \in L_{k-1}$

3) **if**$((l_1[1] = l_2[1]) \wedge (l_1[2] = l_2[2]) \wedge \ldots \wedge (l_1[k-2] = l_2[k-2]) \wedge (l_1[k-1] = l_2[k-1]))${

4) $c = l_1$ *JOIN* l_2; //slučovací krok

5) **if**(obsahuje_nefrekventovanou_podmnožinu(c, L_{k-1}))

6) odstraň *c*; //ostranění nefrekventovaných kandidátů-vzůučovací krok

7) **else** přidej c do C_k;

8) }

9) **return** C_k;

Tato metoda slouží pro generování nových kandidátů. Skládá se ze dvou základních kroků (*slučovací a vylučovací*) . Slučovací krok vytváří z výsledků předchozího kroku nové kandidáty pro frekventované k-množiny. Vstupem do toho kroku jsou dvě lexikograficky seřazené množiny(k-1), které se liší pouze v jednom prvku. Vstupem jsou tedy množiny L_1={p[1],p[2],...,p[k-1]} a L_1={q[1],q[2],...,q[k-1]} , kde jsou všechny prvky shodné a pouze p[k-1] ≠ q[k-1]. Výsledkem je tedy nová množina, taktéž lexikograficky seřazená složená { p[1],p[2],...,p[k-1],q[k-1]}

Druhý krok vylučovací, vstupem do tohoto kroku je nově vytvořená množina po slučovacím kroku. Z výsledné množiny všech k-množin jsou odstraněny ty množiny, jejichž některá k-1 podmnožina se nevyskytuje ve frekventovaných k-1 množinách. Tento krok vychází z tzv. Apriori podmínky, která nám říká, že podpora jakékoliv množiny nemůže být větší, než podpora jakékoliv její podmnožiny. Při tomto kroku nemusíme procházet databází D, ale pouze frekventované množiny a jejich podmnožiny, což značně z efektivní práci algoritmu.

Generování asociačních pravidel z frekventovaných množin

Posledním krokem po provedením algoritmu apriori je ze získaných frekventovaných množin vygenerování asociačních pravidel. Toto generování se provádí na základě rovnice pro výpočet spolehlivosti:

$$conf\,(A \Rightarrow B) = P(B \mid A) = s(A \cup B) / s(A)$$

Za pomocí této rovnice se vygenerují asociační pravidla v následujících fázích:

1) Nejprve se vygenerují z frekventované množiny- *l* všechny její neprázdné podmnožiny

2) Pro každou podmnožinu - s se vygeneruje pravidlo $s \Rightarrow (l-s)$, pro které se vypočítá jeho spolehlivost. Nesplňuje li výsledná spolehlivost podmínku minimální spolehlivosti, jedná se o slabé pravidlo a dále se neuvažuje.

Jelikož jsou asociační pravidla generována z frekventovaných množin je pro ně automaticky splněna podmínka minimální podpory a nemusí se tedy znova počítat.

Zlepšení efektivity apriori algoritmu:

Existuje několik postupů, kterými se dá práce algoritmu apriori ještě více zlepšit. Několik takových postupů je uvedeno níže:

Hashování: Pomocí této metody se dá redukovat počet kandidátských množin položek *Ck (pro k > 1)*. Procházíme li databázi, když zjišťujeme počet výskytů množin položek z *C1*, můžeme z každé transakce generovat všechny množiny položek 2 kandidáty a ty uložíme do hash-tabulky. Při tom si uložíme počet množin uložených pod stejnou hodnotou hashovaní funkce. Jsou li množiny položek s hash-hodnotou menší než je požadovaná minimální hodnota, tak se tyto hodnoty dále neuvažují a mohou se vynechat.

Redukce prohledávaných transakcí: Transakce, která neobsahuje žádnou frekventovanou množinu – k, nemůže obsahovat ani frekventovanou množinu – *k+1*. Taková transakce se buď to z databáze vymaže, nebo nějak vhodně označit, jelikož je pro generování frekventovaných množin nepotřebná a při průchodu databází a jejím načítání by zbytečně zabírala čas.

Rozdělení dat: Tento postup spočívá v počátečním rozdělením celkového objemu dat na několik *N* částí o velikosti *n.* Tato velikost se volí tak, aby právě jedna část o velikosti *n* se vešla celá do paměti. V jednotlivých částech se pak hledají *lokální frekventované množiny*. Pro tyto lokální množiny je nutné, taky upravit míru minimální hodnoty podpory(četnosti) a to podle vzorce minP_L = n * minP. Všechny lokálně nalezené frekventované množiny tvoří kandidáty na *globální* frekventované množiny. Všechny frekventované množiny se naleznou pouze dvěma průchody daty 1. při hledání lokálních frekventovaných množin a 2. při zjišťování frekventovanosti globálních množin.

Vzorkování: Tento postup se uplatní tam, kde je kladen velký důraz na rychlost práce algoritmu, ovšem za cenu méně přesných dat a možností, že se neobjeví veškeré frekventované množiny. Postup

spočívá v tom, že se vybere *n* náhodných transakcí z celé množiny dat tak, aby se všechny vešli do paměti. Frekventované množiny se pak hledají na této množině transakcí.

Metoda FP – stromu: Tento postup se používá k tomu, aby se předešlo problému, který vzniká u generování kandidátu, kterých bývá velké množství. Postup je následující: první fází je komprese databáze reprezentující frekventované položky do struktury nazvané FP – strom. Tento strom se pak rozdělí do podmíněných FP – stromů, které jsou vytvořeny pro každou frekventovanou položku. A z těchto stromů jsou následně získávány frekventované množiny.

2.2.2 Další modifikace algoritmu apriori

Algoritmus AprioriTID

Na rozdíl od klasického algoritmu Apriori, který pracuje s fyzickou databází na disku, tento algoritmus používá pro výpočty množinu $\hat{C}_k$, která je obrazem databáze a je uložena v paměti. S fyzickou databází se pracuje pouze jednou a to při vytváření množiny $\hat{C}_1$. To že se pracuje s fyzickou databází pouze jednou značně přispívá k odbourávání časové náročnosti. Položky v množině mají tvar ((TID),$\{X_k\}$), kde X_k je kandidátem na velkou množinu. Po prvním průchodu je množina shodná s databází, pouze jednotlivé položky transakcí jsou nahrazeny jednoprvkovými množinami obsahující právě tyto položky. Pokud transakce neobsahuje žádného kandidáta na velkou množinu, není v množině obsažena a tím rovněž přispívá k zrychlení algoritmu.

Algoritmus Apriori Itemset

Pracuje podobně, jako předcházející algoritmus AprioriTID s obrazem databáze v paměti. Ovšem je odlišný ve způsobu vyhledávání kandidátů v transakci. Algoritmus AprioriTID procházel celou množinu $\hat{C}_k$, aby nalezl přítomnost kandidáta, algoritmus Apriori Itemset si uchovává u každého kandidáta informaci o tom, ve kterých transakcích se vyskytuje. K tomuto účelu slouží bitové pole, kde 1 a 0 udává informace o tom, ve kterých transakcích se kandidát vyskytuje. Díky tomuto postupu se značně zjednoduší počítání podpory, neboť jen stačí sečíst pole, ve kterých je hodnota nastavená na 1. Pracuje se tak s množinou D , která je ve tvaru ({Xk}, bitové pole udávající přítomnost množiny v transakcích).

Algoritmus AprioriTIDList

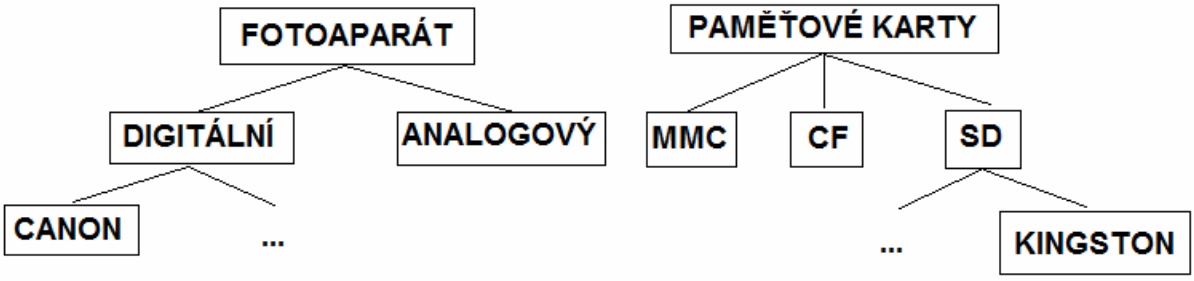
Tento algoritmus je jakýmsi kompromisem mezi oběma dříve zmíněnými algoritmy AprioriTID a Apriori Itemset. Kompromisem je zde míněno to, že je z obou algoritmu vybráno jen to nejlepší čili jejich kladné vlastnosti. Z algoritmu AprioriTID je to zmenšování databáze během výpočtu a užití bitového pole a logického součinu při zjišťování přítomnosti dané množiny v transakcích z algoritmu Apriori Itemset.

2.3 Více úrovňová asociační pravidla

V této sekci se okrajově také zmíníme o více úrovňových asociačních pravidlech, které pracují s položkami na více úrovních abstrakce.

Někdy může být velice obtížné získávat asociační pravidla zvláště pokud máme v systému mnoho položek. Rozšíříme li naše pozorování o určitý stupeň abstrakce, můžeme tak z těchto dat získat zajímavé znalosti. Jedná se například o pravidla typu: koupí li si zákazník fotoaparát CANON s 10px snímačem a 12x zoom a zároveň si koupí Paměťovou kartu SD KINGSTON 4GB, toto pravidlo bude mít s největší pravděpodobností malou podporu, ovšem podíváme li se na toto pravidlo z vyššího stupně abstrakce a vneseme jej do systému, můžeme získat například pravidlo: Koupí li si zákazník fotoaparát(cena > 12000) , koupí si i zároveň paměťovou kartu. A toto pravidlo už může mít určitě mnohem větší podporu. Proto je jednodušší najít asociační pravidla nad více úrovněmi abstrakce.

Pro reprezentaci dat se používá tzv. konceptuální hierarchie. Tato hierarchie se dosti podobá dědičnosti. Položky na nižší úrovni mapují koncepty na vyšší úrovni. Informace o hierarchii se pak musí uchovávat u každé položky v transakci. U každé položky se pak ukládají všichni předchůdci celé hierarchie. Tuto konceptuální hierarchii znázorňuje obr. 2.3.



Obr. 2.3. – Konceptuální hierarchie

2.3.1 Formální definice víceúrovňových asociačních pravidel

Jedná se stále o implikaci tvaru X ⇒ Y, dochází však k některým změnám a to především, že X a Y jsou disjunktní množiny položek, tzn. že žádná položka obsažená v množině Y nesmí být předkem nějaké položky množiny X a to proto, že taková pravidla by měla podporu 100%. Podpora a spolehlivost se počítá stejně jako u jednoúrovňových pravidel. Jako výskyt položky v transakci je považováno to, zda je položka sama vyskytuje, nebo nějaký její předchůdce v celé hierarchii. Konceptuální hierarchie je velmi užitečná, ale jejich použití může způsobit získání redundantních, nebo neužitečných pravidel.

Redundantní asociační pravidla

Pro tyto pravidla platí, že je možné ho přímo odvodit z pravidla, které již bylo získáno na vyšší úrovni. Tedy všechny hledané položky pravidla už jsou na vyšší, nebo stejné úrovni. Tak například existuje li pravidlo typu *fotoaparát ⇒ paměťová karta*, pak pravidlo typu:

Canon PowerShot A570 ⇒ Paměťová karta, má nižší podporu a nepřináší nám, tak žádný další poznatek.

2.4 Multidimenzionální asociační pravidla

Dalším typem asociačních pravidel jsou tzv. multidimenzinální asociační pravidla.

2.4.1 Formální definice multidimenionálních asociačních pravidel

Asociační pravidla, v nichž se predikáty neopakují nazveme mezidimenzionální asociační pravidla. Pokud se predikáty opakují, pak mluvíme i o hybridně dimenzionálních pravidlech. Jde například o pravidlo:

$$\text{Věk}(30...39) \wedge \text{Koupí}(\text{fotoaparát}) \Rightarrow \text{Koupí}(\text{SD paměťovou kartu})$$

Jsou dva základní typy atributů relačních databází. Mohou být buď to Kategorické, nebo Kvantitativní. Z toho Kategorické mají konečný počet hodnot a nemohou být mezi sebou porovnávány(např. zaměstnání, barva). Někdy se tyto hodnoty označují jako nominální, neboť jejich

hodnoty jsou „jména věcí“. Kvantitativní atributy mají zpravidla nekonečný počet možných hodnot, nad kterými je definováno uspořádání.

Jsou tři základní postupy, podle kterých mohou být multidimenzionální asociační pravidla kategorizovány ohledně úpravy kvantitativních atributů.

V prvním postupu jsou kvantitativní atributy kategorizovány s využitím *statické diskreditace*. Jedná se například o rozdělení atributu *váha* na intervaly 40…49, 50…59, atd. Takto diskretizovaný atribut s těmito intervaly může být nyní považován za kategorický.

Ve druhém postupu, jsou kvantitativní atributy diskretizovány do tzv. *segmentů* založených na rozložení dat. Tyto segmenty mohou být dále kombinovány během dolovacího procesu. Diskretizační proces je dynamický, tak aby splňoval určité kriterium, jako je např. maximální spolehlivost pravidla.

Ve třetím postupu, jsou data diskretizovány podle *sémantického významu intervalových dat*. Tento postup bere v potaz vzdálenosti mezi datovými body. Takto získaná asociační pravidla jsou potom označovány, jako *asociační pravidla založená na vzdálenosti*.

2.4.2 Získávání multidimenzionálních asociačních pravidel s využitím statické diskretizace kvantitativních atributů

Při získávání multidimenzionálních asociačních pravidel jsou data diskretizována ještě před procesem zpracování. Kvantitativní data jsou nahrazeny intervaly a kategorická data mohou být nahrazena daty na vyšším stupni konceptuální úrovně. Takto upravená a uložená data v databázi je možné zpracovat za pomocí upraveného algoritmu Apriori.

Získávání kvantitativních asociačních pravidel

Při získávání kvantitativních asociačních pravidel jsou numerické atributy dynamicky při dolovacím procesu nahrazovány intervaly, tak aby vždy vyhovovaly určitým požadavkům, například větší spolehlivost. Máme li například pravidlo $A_{quan1} \wedge A_{quan2} \Rightarrow A_{cat}$, kde na levé straně jsou 2 kvantitativní atributy a na straně pravé 1 kategorický atribut. Takovéto pravidlo se nazývá dvojdimenzionální. Atributy na levé straně jsou testovány na rozsahy kvantitativních atributů. Pro získávání takovýchto pravidel se používá metoda ARCS(association rule clustering system). Tato metoda mapuje kvantitativní atributy do 2-D mřížky podle podmínky dané kategorickým atributem.

Následně se prohledává mřížka a zjišťují se shluky bodů, ze kterých se pak generují asociační pravidla. Tato metoda zahrnuje následující kroky:

Rozdělení do segmentů: Kvantitativní atributy mohou nabývat velkých rozsahů. Abychom snížili výslednou velikost používáme místo konkrétních hodnot intervaly kvantitativních atributů a tyto intervaly jsou nazvány segmenty. Pro rozdělení do segmentů rozeznáváme 3 základní strategie segmentace.

1) *Rozdělení se stejnou velikosti intervalů pro každý segment*.
2) *Rozdělení se stejnou hloubkou segmentu*(každý segment má přibližně stejný počet položek, které jsou do něj přiřazeny)
3) *Uniformní rozdělení záznamu uvnitř segmentu.*

Nalezení frekventovaných množin. Po vynesení počtu výskytů pro každou kategorii do 2 – D mřížky, můžeme toto pole projít a získat tak frekventované množiny splňující minimální podporu a minimální spolehlivost. Silná asociační pravidla se pak generují z těchto hodnot.

3 XML

3.1 Úvod

XML je otevřený standart pro definování vlastního strukturovaného jazyka, představujícího formát pro ukládání dat. Tuto definici představuje XML schéma, nebo DTD soubor, který se pak používá pro ověření správnosti zpracovaného dokumentu. Hlavními výhodami uložených dat v XML je, že těží z výhod otevřených standardů a umožňují snadné výměny mezi aplikacemi. XML dokument je také možné reprezentovat pomocí stromové struktury, která umožňuje členit dokument na jednotlivé části a poskytuje další užitečné informace.

XML a jmenné prostory umožňují strukturovat dokumenty s použitím značek, neboli tagů. Dokumenty vyhovující specifikacím XML a jemných prostorů, se mnohou skládat z různých syntaktických prvků, jako jsou například elementy, atributy, deklarace jmenných prostorů, instrukce ke zpracování, komentáře a text. Podle [6] a [7].

3.2 Elementy

Elementy obvykle tvoří většinu obsahu XML dokumentu. Každý XML dokument obsahuje právě jeden kořenový, neboli element dokumentu nejvyšší úrovně. Každý element obsahuje jméno, příčemž se jednotlivé elementy nemusí deklarovat a programátor si tak může vymyslet jméno elementu jaké chce. Dále element může obsahovat potomky, což jsou další elementy, instrukce ke zpracování, komentář, text, nebo znaky, na nižším stupni hierarchie. Elementu potomek se také říká dítě, nebo dětský potomek. Dětské elementy jsou uspořádány a záleží tak na jejich pořadí. Elementy mohou být opatřeny atributy, u kterých nezáleží na pořadí.

Element je tvořen počáteční značkou a koncovou značkou. Syntaxe obou značek je následující: Počáteční značka obsahuje dvojici ostrých závorek (< a >) ,mezi kterými je ohraničeno jméno elementu. Jelikož je element párový tag, musí být jeho obsah ukončen koncovou značkou, která má za počátečním znakem lomítko(</), následuje jméno elementu a koncová závorka(>). Pokud element neobsahuje žádná data, je možné použít zkrácený zápis jeho příklad je uveden níže.

Příklad dětského elementu:

```
<element>
    <detsky_element>text</detsky_elemetn>
</element>
```

Prázdný element:

```
<element></element>
```

Zkrácený zápis:

```
<element/>
```

3.3 Atributy

Elementy mohou obsahovat atributy. Ty lze použít pro uložení vlastních dat, nebo mohou obsahovat metadata o elementu, neboli další informace o elementu, ve kterém se vyskytují. Atributy jsou uloženy v počáteční značce elementu a jsou tvořeny jménem a hodnotou odděleny od sebe znakem '='. Pro jména atributů platí stejná pravidla, jako pro jména elementů. Hodnoty jsou textové řetězce, které musí být uzavřeny do uvozovek či apostrofů. Atributů v jednom elementu může být libovolné množství, ale nesmí se jmenovat stejně.

Příklad datového atributu:

```
<osoba tlak="Normální"  Věk="33">
```

Příklad atributu s metadaty:

```
<výška jednotky ="cm">180</výška>
```

3.4 Jmenné prostory

Vzhledem k tomu, že je možné zvolit si svá vlastní jména elementů, mohou si návrháři pro některé elementy zvolit stejná jména. Jmenné prostory(namespace) přinášejí způsob, jak od sebe odlišit stejné lokální jméno, ale pocházejí z odlišných slovníků. Toho se dosáhne pokud spojíme jméno elementu s jmenným prostorem. Jmenný prostor má jméno utvořené dle specifikace URI, které slouží jako unikátní řetězec. Jméno jmenného prostoru a lokální jméno elementu tvoří dohromady globální unikátní jméno, kterému se říká kvalifikované jméno.

Deklarace jmenného prostoru se provádí uvnitř počáteční značky. Tato Deklarace má syntaxi je xmlns:prefix="URI", je také možné deklarovat výchozí jmenný prostor, který se aplikuje na všechny elementy bez prefixu. Jeho syntaxe je pak následující xmlns="URI". Každý jmenný prostor má rozsah platnosti, do kterého patří element, ve kterém je jmenný prostor deklarován a všichni jeho potomci.

Příklad:

```
<pre: Osoba xmlns:pre="urn:example-org:Lidé">
    <jméno>Matěj</jméno>
    <výška>180</výška>
</pre:Osoba>
```

3.5 Deklarace XML

Hned na začátku XML dokumentu musí být deklarace XML. Tato deklarace se skládá až ze tří dvojic, pro která platí stejná pravidla jako pro atributy. Jediným povinným atributem je atribut *version* další atributy *encoding* a *standalone* jsou nepovinné, ale jejich pořadí v rámci deklarace je závazné.

Deklarace XML dokumentu začíná posloupností znaků <?xml a končí posloupností znaků ?>. Všechny deklarace XML dokumentu musí obsahovat atribut *version* s hodnotou 1.0. Znaková sada použitá v dokumentu se pak deklaruje v atributu *encoding*. XML data jsou ze své podstaty Unicode, i když jsou uloženy v jiém kódování než Unicode. Pokud se XML neodkazuje na žádné externí zdroje, řekneme, že je samostatný (standalone) a přidáme do deklarace atribut *standalone* s hodnotou *yes*.

Příklady:

<?xml version=“1.0“?>

<?xml version=“1.0“ encoding=“UTF-8“?>

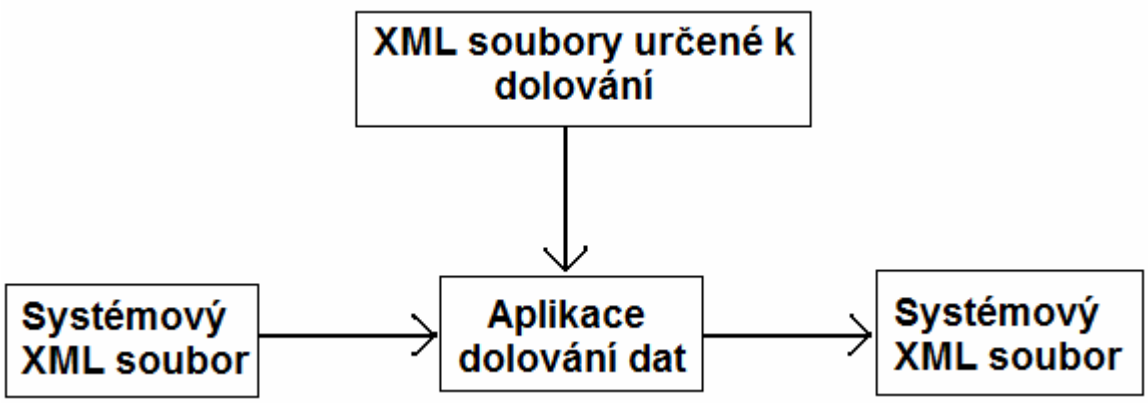
<?xml version=“1.0“ encoding=“UTF-16“ standalone=“yes“?>

4 Návrh koncepce aplikace

4.1 Úvod

Cílem této práce bylo implementovat jednu vybranou metodu získávání znalostí z databází. V tomto případě je zvolenou metodou pro získávání znalostí, metoda pro generování asociačních pravidel. Při návrhu aplikace jsem vycházel ze získaných znalostí o komplexním projektu získávání znalostí. Přičemž tato aplikace je jen jednou z částí celého systému a to konkrétně modulu dolování dat. Dolování dat z XML dokumentu je odlišný způsob od dolování dat z databází a to především proto, že zde nelze použít SQL dotazy, nýbrž jiný sofistikovanější přístup. Návrh koncepce této aplikace je znázorněn na obrázku 3.1.

Vstupem do aplikace jsou XML soubory. Jeden XML soubor je nazýván systémový (konfigurační). Tento soubor obsahuje potřebné informace pro správné nastavení a následné spuštění aplikace. Těmito informacemi jsou například názvy asociovaných elementů, minimální četnost, minimální spolehlivost atd.. Dalšími XML soubory jsou samotné záznamy, nad kterými se bude provádět získávání znalostí. Aplikace je po získání těchto informací spuštěna a je zahájen proces dolování dat pomocí upraveného algoritmu Apriori. Výsledkem po ukončení aplikace je nový XML soubor, který obsahuje informace obsažené v systémovém souboru a vygenerovaná pravidla.



Obr. 3.1 – Koncepce aplikace

4.2 Formát dat v XML souboru(databázi)

Data uložené v XML souboru určeném ke zpracování implementovanou metodou jsou již po provedeném předchozím kroku předzpracování dat. Data jsou vyčištěna od přebytečných dat a šumů. Data jsou uložena v jednom kořenovém elementu a všechny záznamy obsahují úplná data. Data, která jsou numerická a tudíž vhodná pro rozdělení dat do intervalů, nejsou přiřazena k intervalů, před

zpracováním dokumentu aplikací. Přiřazení se provádí při běhu aplikace při načítání dat ze souboru a ukládání do paměti. Tedy pokud záznam obsahuje numerický záznam je načtena jeho nominální hodnota a posléze je převedena na interval odpovídající rozsahu, do kterého prvek spadá a takto je uložen v paměti. Původní hodnota se v načteném souboru nemění.

4.3 Formát systémového XML souboru

Jelikož je tato aplikace spouštěna samostatně a není součásti celého systému, musí být vstupní dokument vytvořen ručně. V případě, že by aplikace byla součástí celého systému byl by tento soubor vytvořen a předán modulem pro předzpracování dat. Proto je nutné definovat obsah tohoto souboru.

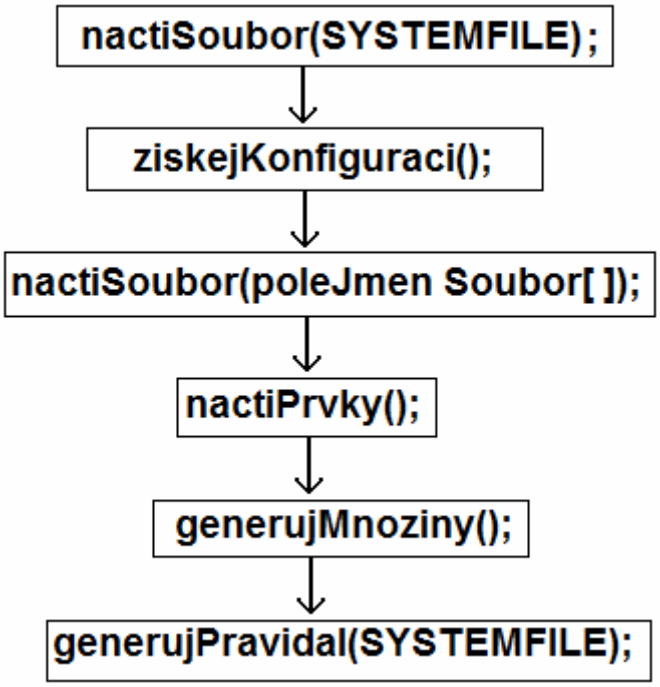
V tomto dokumentu je nutné uvést elementy, které se mají asociovat, jejich počet není nějak omezen. Pro snažší orientaci a úpravy v datech, jsou zde uvedeny elementy, které jsou číselné a také maximální vzdálenost kvantitativních hodnot pro rozdělení do intervalů. Nezbytným údajem je také cesta, kde se XML soubory nacházejí. Dále je nutné uvést element obsahující minimální četnost, minimální spolehlivost a maximální hloubku zanoření. Dále aplikace předpokládá, že daný systémový soubor obsahuje vytvořený element *<rules>*, který bude prázdný a po ukončení aplikace bude obsahovat vygenerovaná pravidla.

5 Implementace aplikace

5.1 Úvod

Jak už bylo uvedeno v předcházející kapitole byl pro implementaci této aplikace vybrán objektově orientovaný programovací jazyk Java ve verzi 1.6. Mezi jeho přednosti patří také velké množství knihoven pro práci s XML soubory. Bylo nutné vybrat, kterou knihovnu pro zpracování XML souboru vybrat. Pro velké rozšíření a snadnou manipulaci jsem zvolil knihovnu DOM (*Document object model*). Na rozdíl například od knihovny SAX parser DOM načte celý dokument do paměti a vytvoří tam stromovou objektovou reprezentaci, a tak je možné provádět změny, či vytvářet nový XML soubor.

Celá aplikace se dá rozdělit na několik problémů, které bylo nutno vyřešit a následně sjednotit v jeden celek. Prvním problémem je načtení konfiguračního (systémového) XML souboru. Z tohoto dokumentu je nutné získat informace pro správné nastavení aplikace. Jako druhý problém je potřeba vyřešit nalezení XML souborů určených pro dolování a načtení požadovaných elementů na základě informací získaných z konfiguračního souboru. Dalším problémem je samotné dolování dat, hledání frekventovaných množin a v neposlední řadě je to samotné vygenerování asociačních pravidel do konfiguračního souboru. Obrázek č.4.1 ukazuje abstraktní pohled na metody, které řeší jednotlivé problémy a jejich návaznosti.



Obr. 4.1 Návaznost metod

5.2 Proces zpracování konfiguračního souboru

Jak již bylo uvedeno konfigurační soubor pro tuto aplikaci musí být vytvořen ručně. Pokud by byl modul dolování dat součástí celého systému, předal by konfigurační soubor společně s upravenými daty modul pro předzpracování dat. Načtení tohoto souboru je použit standart připravený konsorciem W3C, jedná se o DOM (document object model), více o něm v [3].

Před samotným zpracováním musím nejprve znát popis konfiguračního souboru. Tento souboru musí zaručeně obsahovat tyto elementy: <*associate*> potomky tohoto elementu jsou elementy, které udávají výpis prvků (elementů), které se budou v XML dokumentech dolovat a budou tak z nich generovány frekventované množiny. Dalším nezbytným elementem je <*number*>, tento element obsahuje potomky, kteří udávají numerické prvky. To je důležité pro samotné dolování dat, kdy se načítají a upravují prvky z XML dokumentů. Význam bude vysvětlen později. Dalším elementem je <*search*>, obsahuje li databáze více záznamů (tabulek), tak tento element určí, ve které se bude dolování provádět. Element <*path*> určuje cestu k XML dokumentům, nad kterými se bude provádět dolování. Element <*quantity*> vymezuje rozpětí intervalu kvantitativních prvků. A nakonec elementy <*minSupp*>, <*minConf*>, <*maxDepth*> určují požadovanou minimální četnost (podporu), minimální spolehlivost pro vygenerovaná pravidla a maximální hloubku zanoření ve frekventovaných množinách. Maximální zanoření by nemělo být větší, než je počet asociovaných prvků.

Pro načtení XML souboru slouží metoda *nactiSouvor()* , který využívá pro vytvoření objektů pro parser *DocumentBuilderFactory* a *DocumentBuilder*. Jako parametry tato metoda dostane seznam jmen souboru, které se mají zpracovat společně s cestou, kde se uvedené soubory nacházejí. Pro sofistikovaný průchod načteným stromem dokumentu se v metodě *getAsociate()* používá rozhraní *NodeIterator*, které prochází uzly v načteném seznamu lineárně od prvního k poslednímu. Tato metoda na základě zpracovaného dokumentu načte data, která se ukládají do globálních proměnných, které se posléze použijí k nastavení konfigurace samotného dolování dat. Metoda *nactiSouvor()* je shodná jak pro zpracování konfiguračního souboru tak pro zpracování datových XML souborů, liší se pouze cesta k nim. Ukázka konfiguračního (systémového) souboru, příloha [A].

5.3 Proces dolování dat z XML souborů

Jakmile máme zpracovaný konfigurační soubor a víme jaké elementy se budou asociovat můžeme začít se zpracováním XML souborů s daty. Nejprve je nutné soubory zpracovat a to pomoci metody *nactiSoubor(),* podobně jako u načítání konfiguračního souboru. V tomto případě je rozdíl v tom, že v cyklu zpracováváme všechny dokumenty, kterých může být více, a okamžitě z nich čteme prvky (elementy), které se mají asociovat. Tyto prvky ukládáme do globálního vektoru prvků. Pro každý typ elementu je vytvořen jeden vektor, do kterého se ukládají elementy stejného typu. Pro každý

prvek je vytvořena samostatná třída *Prvek()*, která obsahuje dvě vlastní proměnné, *data*- hodnota elementu (popřípadě vektor hodnot pro další frekventovanou množinu) a *supp* – proměnná, do které je uložena četnost prvku či množiny. Při načítání prvku je potřeba rozlišovat, zda li je prvek kvantitativní, nebo kategorický. Kdyby byl modul součástí celého systému, měl by tuto práci na starosti modul předzpracování dat. Je li prvek kategorický nic se pro něj nemění a je uložený do příslušné struktury. Ovšem je li prvek kvantitativní, musí být před uložením ještě zpracován. Aby se dosáhlo toho, že prvek bude mít větší podporu jsou kvantitativní prvky transformovány na příslušné intervaly hodnot, které si jsou blízké. Intervaly mají definovanou šířku, která je načtena z konfiguračního souboru, tak např. šířka 10 tzn., že prvky jejichž hodnota je v řádu jednotek od 0 do 9 si jsou rovny při počítání např. četnosti (podpory).

Dalším krokem po načtení a upravení hodnot z dokumentů do paměti je vypočítání četnosti (podpory) těchto prvků. Tuto činnost má na starosti metoda *getFirstSupport ()* . Jelikož XML soubory obsahují velké množství dat, tak aby se předešlo velkým paměťovým náročnostem jsou prvky při počítání četnosti porovnávány opětovným postupným procházením všech XML souborů. Po vypočítání četností se zavolá metoda *deleteMinSupp()*, která má jako parametr vektor prvků a požadovanou minimální četnost, která byla načtena z konfiguračního souboru. Na základě těchto informací odstraní z vektoru prvků všechny prvky nesplňující minimální četnost.

Jsou li nežádoucí prvky vymazány, může začít samotné generování frekventovaných množin pro asociační pravidla, podrobně tento postup byl vysvětlen v kapitole *Asociační pravidla*, proto se zaměříme spíše na praktickou část problému. Pro vygenerování prvních kandidátů pro frekventované množiny slouží metoda *generateFirstCandidats()*, vstupem jsou prvky s vypočítanou četností. Tato metoda vygeneruje všechny kombinace vstupních prvků, např. jsou li na vstupu 3 vektory reprezentující 3 druhy prvků A, B, C metoda vygeneruje množiny typu [A, B], [A, C], [B, C]. Poté je u těchto množin vypočítána četnost obdobnou metodou, jako u počítání četnosti prvků. U množin je dále porovnávána četnost a minimální četností, pokud není podmínka minimální četnosti splněna je množina nefrekventovaná a je uložena do vektoru nefrekventovaných množin a odstraněna z vektoru kandidátů. Tímto způsobem jsou vygenerovány frekventované množiny. Jelikož je odlišný způsob uložení prvků a frekventovaných množin jsou i odlišné metody pro generování víceprvkových množin, které generuje metoda *generateCandidats()*.Tato metoda pracuje v cyklu, dokud nejsou vygenerovány všechny frekventované množiny, nebo dokud není dosaženo maximální hloubky zanoření, jejíž hodnota byla načtena z konfiguračního souboru. Metoda vygeneruje nové kandidáty na frekventované množiny, ještě před počítáním četnosti jsou z nových kandidátů odstraněny množiny, které obsahují nefrekventované podmnožiny. Tento krok značně urychluje práci, protože se nemusí počítat četnost pro všechny, tedy i nefrekventované, množiny. Poté je vypočítána četnost metodou *support()* a nevyhovující množiny se odstraní z vektoru kandidátů. Výsledkem těchto metod jsou frekventované množiny jejž ukázka je v kapitole *Testování na reálných datech*.

5.4 Proces generování asociačních pravidel

Jako poslední krokem zůstává samotné generování asociačních pravidel do systémového souboru. Generování pravidel se provádí po každém vygenerování nových kandidátů. Tentoto krok má na starosti Metoda *generateAsociate()* , která dostane jako parametry nové kandidáty na pravidla a datový objekt výstupního souboru. Generování pravidel je zde zjednodušeno pouze na pravidla, kdy jsou všechny prvky na levé straně, kromě jednoho, které je na pravé straně tedy např. pravidlo:

$$jmeno=Jan \wedge vaha=80...89 \wedge \ vyska=170...179 \Rightarrow vek=60...69$$

Z frekventované množiny jsou vygenerovány všechny takovéto možné kombinace, pro které je vždy spočítána spolehlivost metodou *countConfidence()* , která pro počítání využívá četnosti podmnožiny, která vznikal na levé straně a četnosti celé množiny. Tento postup značně urychluje průběh aplikace na rozdíl od toho kdyby se muselo procházet znovu několik XML souboru. Po vypočítání četnosti jsou uložena do paměti pravidla splňující podmínku minimální spolehlivosti, jelikož jsou pravidla generována z frekventovaných množin je pro ně podmínka minimální podpory automaticky splněna. Po vytvoření pravidel metoda writeDoc() zapíše pravidla do systémového XML souboru za pomocí tříd *TransformerFactory* a *Transformer*. Ukázka výstupního souboru je v příloze [B].

5.5 Spuštění aplikace

Pro bez chybové spuštění aplikace je potřeba zadat cestu a název systémového (konfiguračního) souboru, který se nachází v adresáři /Apriori/System File/. Jako vývojové prostředí byla zvolena aplikace NetBeans IDE 5.5, která kompiluje celý projekt přímo do archivu jar, a proto jej lze spustit dvěma způsoby. Buď to přímo zpuštěním archivu jar, nebo kompilací hlavní třídy Apriori a tu poté spustit. Po spuštění aplikace se načte systémový soubor a zpracují se XML datové soubory. Po vygenerování pravidel vznikne nový soubor *systém-kdd.xml* v adresáři, kde se nachází i konfigurační soubor, jenž je výstupem této aplikace. Pokud proběhne vše v pořádku, jsou v souboru uložena vygenerovaná pravidla společně s informacemi pro konfiguraci aplikace.

6 Testování na reálných datech

Testovací soubory XML jsou tvořeny záznamy o pacientech, které byly uloženy jako tabulka do XML souborů a nad kterými bylo prováděno samotné dolování. Bylo provedeno několik experimentálních testů s různými elementy pro dolování a různou konfigurací systémového XML souboru. Bylo získáno několik zajímavých znalostí. Dva z testů, které byly provedeny jsou uloženy jako Test1 a Test2 na přiloženém CD-ROM mediu. Ukázka konfiguračního souboru je v příloze [A]. Ukázka výstupního XML souboru s vygenerovanými asociačními pravidly je uvedena v příloze [B].

Experimenty byly prováděny nad tabulkou *Pacienti*. Jeden záznam pacienta obsahuje elementy udávající tyto údaje: z řad kategorických jsou to *Jméno, Příjmení, Tlak* a z řad kvantitativních to jsou *váha*, *výška* a *věk*. V každém ze souborů (celkový počet 2) je obsaženo 20 záznamů o jednotlivých pacientech. Což je dostatečný obsah pro provedení testů na získání zajímavých znalostí z těchto dat.

Jako názornou ukázku zde uvedu testy dolování asociačních pravidel nad kombinací kategorických a kvantitativních dat. Jako vhodné kandidáty jsem zvolil z řad kategorických prvků element *tlak* a z kategorie kvantitativních prvků elementy *váha*, *výška* a *věk*. Po ukončení aplikace budou tedy vygenerovány tří-dimenzionální pravidla obsahující 3 prvky na levé straně a jeden prvek na straně pravé. Pro kategorická data bylo pro přiřazení prvků do intervalů zvolena maximální vzdálenost hodnot 10, což by mělo zaručit vyhledání prvků, které si jsou hodnotami velmi blízko. Pro nastavení celého procesu dolování, pak byly zvoleny tyto parametry pro *minimální četnost(podporu)* = 2 (udává počet výskytů v XML souborech), pro *minimální spolehlivost* = 60 a pro *maximální hloubku* zanoření ve frekventovaných množinách = 4.

Po spuštění aplikace byly vypočítány četnosti prvků splňující podmínku minimální podpory, jedná se o tyto prvky:

vaha=100...109 : 3	vek=10...19 : 9	vyska=160...169 : 7	tlak=nizky : 8
vaha=50...59 : 6	vek=20...29 : 9	vyska=170...179 : 13	tlak=normal : 21
vaha=60...69 : 4	vek=30...39 : 6	vyska=180...189 : 16	tlak=vysoky : 11
vaha=70...79 : 10	vek=40...49 : 2	vyska=190...199 : 3	
vaha=80...89 : 9	vek=50...59 : 3	----------	----------
vaha=90...99 : 6	vek=60...69 : 2		
----------	vek=70...79 : 2		
	vek=80...89 : 5		
	vek=90...99 : 2		

Takto velkého množství prvků pro generování frekventovaných množin bylo dosaženo zvolením rozsahu vzdáleností hodnot prvků rovno 10. Pokud bychom zvolili rozsah větší například 50, bylo by sice méně prvků, ale za to by nám vzrostla četnost jednotlivých intervalů. Což by se uplatnilo v datech, kde nemají prvky tak blízké kvantitativní hodnoty.

Po vygenerování prvků, se z nich vytvořily frekventované množiny, které opět splňují minimální podporu. Zde uvedu příklad pouze posledních vygenerovaných množin, obsahující všechny hledané prvky.:

1. [tlak=vysoky, vaha=100...109, vek=80...89, vyska=180...189] : 2
2. [tlak=normal, vaha=70...79, vek=50...59, vyska=170...179] : 2
3. [tlak=normal, vaha=80...89, vek=20...29, vyska=180...189] : 2

Z těchto vygenerovaných frekvenčních množin se v poslední fázi vygenerují hledaná asociační pravidla. Zde uvedu příklad vygenerovaných asociačních pravidel pro 1. z uvedených frekventovaných množin. Pro tuto množinu byly vygenerovány do výstupního XML dokumentu 4 asociační pravidla splňující podmínku minimální podpory.

$\{(vaha=100\ldots109) \wedge (vek=80\ldots89) \wedge (vyska=180\ldots189)\} \Rightarrow (tlak=vysoky)$ conf = 100 supp = 2

$\{(tlak=vysoky) \wedge (vek=80\ldots89) \wedge (vyska=180\ldots189)\} \Rightarrow (vaha=100\ldots109)$ conf = 66 supp = 2

$\{(tlak=vysoky) \wedge (vaha=100\ldots109) \wedge (vyska=180\ldots189)\} \Rightarrow (vek=80\ldots89)$ conf = 100 supp = 2

$\{(tlak=vysoky) \wedge (vaha=100\ldots109) \wedge (vek=80\ldots89)\} \Rightarrow (vyska=180\ldots189)$ conf = 100 supp = 2

Z těchto získaných znalostí se dá odvodit skutečnosti, že pokud má pacient váhu mezi 100 až 109 kg, věk mezi 80 až 89 lety a výšku od 180 do 189 cm je velice pravděpodobné, že má také vysoký tlak. Pokud bychom chtěli nalézt jiné znalosti, museli bychom upravit vstupní konfigurační soubor, tak aby vygenerovaná pravidla odpovídala našim požadavkům.

Hlavním úkolem testování však bylo ověřit správnou funkčnost implementované metody, kterou bylo dolování asociačních pravidel za pomocí upraveného algoritmu Apriori. Testy proběhly dle očekávání a plně splnily požadavky. Toto testování leze tedy označit jako úspěšné.

7 Závěr

V této bakalářské práci jsem se seznámil s problematikou získávání znalostí z databázi. Podle zadání jsem se zaměřil na jednu z metod a to konkrétně získávání asociačních pravidel. Použil jsem upravený Apriori algoritmus, modifikovaný tak, aby spolupracoval s XML soubory, nad kterými bylo prováděno samotné dolování. Tento algoritmus je schopný provádět dolování, jak nad kvantitativními, tak i nad kategorickými atributy. Pro konfiguraci aplikace jsem použil ručně vytvořený konfigurační XML soubor, ve kterém se nastavují parametry pro dolovací úlohu.

Úlohy pro dolování dat jsou vysoce náročné na paměť, protože pro získání zajímavých znalostí je potřeba nashromáždit velké množství vstupních dat, i několik XML souborů. Proto jsem při implementaci kladl důraz na sekvenční zpracování úlohy a co nejvíce omezit využité paměti na velikosti zpracovávaných dat. Nevyhneme se však ukládání zpracovaných prvků potřebných ke generování frekventovaných množin a samotné frekventované množiny. Ovšem při dalším zpracování např. počítání podpory, už takový krok není potřeba a ušetříme paměť postupným procházením XML souborů. Z hlediska časové náročnosti je rychlost ovlivněna rychlostí načítáním a zapisováním prvků do XML dokumentu. Jako vylepšení do budoucna bych navrhoval implementaci struktury stromu pro ukládání frekventovaných množin, což by umožnilo rychlejší hledání a nebylo by nutné procházet položky sekvenčně.

Metodu jsem testoval na připravených datech na kterých bylo možno dobře odzkoušet vlastnosti práce algoritmu. První testovací databáze v XML souborech byla vytvořena uměle pro názorné demonstrování funkčnosti aplikace. Tato databáze není nějak rozsáhlá tudíž její zpracování nezabere tolik času a je vhodná pro demonstraci.

Výsledkem této práce je modul pro dolování asociačních pravidel z XML dokumentů. Tento modul by mohl být součástí celého systému, pro získávání znalostí z databází, v tomto případě tedy konkrétně z XML souborů.

Součástí této práce jsou na přiloženém CD-ROM mediu uvedeny dva demonstrační testy, které jsou uloženy ve skožkách Test1 a Test2. Pro jejich správnou instalaci a používání je přiložen u každého soubor INSTAL, ve kterém jsou popsány pokyny pro správnou instalaci a spuštění. Dále je CD – ROMu uložen html soubor README, který obsahuje návod na instalaci a správné používání samotné aplikace.

Literatura

[1] JIAWEI, Han, KAMBER, Micheline. *Data Mining: Concepts and Techniques*. 2nd edition. San Francisco : Morgan Kaufmann, 2006. 743 s. ISBN 10: 1-55860-901-6.

[2] BERKA, Petr. *Dobývání znalostí z databází*. Praha : Academia, 2003. 366 s.
ISBN 80-200-1062-9.

[3] HEROUT, Pavel. *Java a XML*. 1. vyd. České Budějovice : KOPP, 2007. 313 s.
ISBN 978-80-7232-307-4.

[4] ZENDULKA, Jaroslav, BÁRTÍK, Vladimír , LUKÁŠ, Roman, RUDOLFOVÁ, Ivana. *Získávání znalostí z databází*. Fakulta Informačních Technologii VUT Brno, 2006

[5] STRYKA, Lukáš. *Modul Dolování asociačních pravidel*, diplomová práce. Fakulta Informačních Technologii VUT Brno, 2003

[6] PROCHÁZKA, Martin , BLAŤÁK Jan. *Klasifikace XML dokumentů*, Laboratoř vyhledávání znalostí, Fakulta Informatiky, Masarykova Univerzita v Brně

[7] SKONNARD, Aaron, GUDGIN, Martin. *XML pohotová referenční příručka*. 1. vyd. Praha : Grada Publishing, a.s, 2006. 344 s. ISBN 80-2470972-4.

Seznam příloh

Příloha A.

Ukázka vstupního XML konfiguračního souboru.

Konfigurační soubor, ve kterém se nastavují potřebné parametry pro konkrétní dolovací úlohy. Element `<asssociate>` udává elementy ze kterých se budou generovat frekvenční množiny. Element `<number>` Udává, které kvantitativní prvky se budou rozdělovat do intervalů podle elementu `<quantity>`, ten udává maximální vzdálenost hodnot prvků. Element `<path>` udává cestu k XML souborům nad kterými se bude dolovat. Element `<search>` udává, která tabulka se bude vyhledávat. Elementy `<minSupp>`, `<minConf>`, `<maxDepth>` slouží k nastavení minimální četnosti, spolehlivosti a maximální hloubky zanoření.

```
<?xml version="1.0" encoding="windows-1250"?>
<project>
  <asssociate>
    <element>vaha</element>
    <element>vek</element>
    <element>tlak</element>
    <element>vyska</element>
    <element>jmeno</element>
  </asssociate>
  <number>
     <element>vaha</element>
     <element>vek</element>
     <element>vyska</element>
  </number>
  <search>pacient</search>
  <minSupp>2</minSupp>
  <minConf>66</minConf>
  <maxDepth>4</maxDepth>
  <path>/Apriori/file/</path>
  <quantity>10</quantity>
  <rules>
  </rules>
</project>
```

Příloha B

Testování funkčnosti aplikace

Součástí této práce jsou přiloženy vzorové testy, které názorně demonstrují funkčnost aplikace. Nastavení jednoho konfiguračního souboru a ukázka výsledného XML souboru jsou uvedeny níže.

Nastavení konfiguračního(systémového) XML souboru:

```
<project>
  <asssociate>
    <element>vaha</element>
    <element>vek</element>
    <element>tlak</element>
  </asssociate>
  <number>
     <element>vaha</element>
     <element>vek</element>
  </number>
  <search>pacient</search>
  <minSupp>2</minSupp>
  <minConf>70</minConf>
  <maxDepth>4</maxDepth>
  <path>/Andrew/java/Apriori/file/</path>
  <quantity>10</quantity>
  <rules>
  </rules>
</project>
```

Test byl proveden nad vytvořeným záznamem pacientů kombinující v sobě kvantitativní(věk, váha, výška) a kategorická data(jméno, příjmení, tlak).

Ukázka výstupního XML dokumentu je znázorněna níže. Výstup je generován do elementu <rules> ve vstupním dokumentu. U každého vygenerovaného pravidla jsou jako atributy uloženy hodnoty vypočítané četnosti(četnost 2 odpovídá 22% podpory) a spolehlivosti pravidla.

```
<project>
  <asssociate>
    <element>vaha</element>
    <element>vek</element>
    <element>tlak</element>
  </asssociate>
  <number>
     <element>vaha</element>
     <element>vek</element>
  </number>
  <search>pacient</search>
  <minSupp>2</minSupp>
  <minConf>70</minConf>
  <maxDepth>4</maxDepth>
  <path>/Andrew/java/Apriori/file/</path>
  <quantity>10</quantity>
  <rules>
  <rule confident="100" support="2">
      <left>
        <item name="vaha">90...99</item>
      </left>
      <right>
        <item name="vek">20...29</item>
      </right>
    </rule>
    <rule confident="75" support="3">
      <left>
        <item name="vek">60...69</item>
      </left>
      <right>
        <item name="tlak">normal</item>
      </right>
    </rule>
    <rule confident="100" support="2">
      <left>
        <item name="vaha">70...79</item>
        <item name="vek">30...39</item>
```

```
      </left>
      <right>
        <item name="tlak">nizky</item>
      </right>
    </rule>
    <rule confident="100" support="2">
      <left>
        <item name="tlak">nizky</item>
        <item name="vek">30...39</item>
      </left>
      <right>
        <item name="vaha">70...79</item>
      </right>
    </rule>
    <rule confident="100" support="2">
      <left>
        <item name="tlak">nizky</item>
        <item name="vaha">70...79</item>
      </left>
      <right>
        <item name="vek">30...39</item>
      </right>
    </rule>
    <rule confident="100" support="2">
      <left>
        <item name="vaha">80...89</item>
        <item name="vek">60...69</item>
      </left>
      <right>
        <item name="tlak">normal</item>
      </right>
    </rule>
    </rules>
</project>
```